Efficient Abstraction of Clock Synchronization at the Operating System Level

Alessandro Sorrentino
DEIB, Polytechnic University of Milano, Italy

Federico Terraneo
DEIB, Polytechnic University of Milano, Italy

Alberto Leva
DEIB, Polytechnic University of Milano, Italy

Abstract

Distributed embedded systems are emerging and gaining importance in various domains, including industrial control applications where time determinism – hence network clock synchronization – is fundamental. In modern applications, moreover, this core functionality is required by many different software components, from OS kernel and radio stack up to applications. An abstraction layer devoted to handling time needs therefore introducing, and to encapsulate time corrections at the lowest possible level, the said layer should take the form of a timer device driver offering a *Virtual Clock* to the entire system. In this paper we show that doing so introduces a nonlinearity in the dynamics of the clock, and we design a controller based on feedback linearization to handle the issue. To put the idea to work, we extend the Miosix RTOS with a generic interface allowing to implement virtual clocks, including the newly designed controller that we call FLOPSYNC-3 after its ancestor. Also, we introduce the resulting virtual clock in the TDMH [20] real-time wireless mesh protocol.

2012 ACM Subject Classification Computer systems organization → Real-time operating systems; Networks → Time synchronization protocols; Computer systems organization → Embedded software

Keywords and phrases Clock synchronization, Real-time operating systems, Embedded software, Real-time control

Digital Object Identifier 10.4230/OASIcs.NG-RES.2023.4

1 Introduction

The world of embedded systems is evolving from isolated to distributed systems. This move can be observed in several research and market trends such as the Industrial Internet of Things (IIoT) [16]. As a result, clock synchronization is becoming a key technology to enable both real-time industrial applications as well as low energy wireless protocols [23]. At the application level, synchronization in distributed embedded systems allows the execution of coordinated tasks among multiple devices [13], allows to perform sensing and reconstruction of spatially distributed phenomena [10, 2], while the availability of synchronization at the network level enables the use of TDMA protocols [1, 20, 7], being thus fundamental for real-time communication among devices. Since in modern embedded operating systems both applications and OS components – such as the radio stack – can benefit from clock synchronization, an abstraction layer that handles time correction directly at the OS level is therefore needed. Moreover, from a software engineering perspective, the presence in the OS codebase of both corrected and uncorrected time values is a potential source of errors. Therefore, it becomes desirable to encapsulate time correction at the lowest possible level, such as the timer device driver.

However, using corrected times in the entire OS codebase introduces an issue: the uncorrected time is usually required by the clock synchronization algorithm itself. Efficient clock synchronization schemes such as FLOPSYNC-2[22] require uncorrected timestamps of received synchronization packets, and performing clock synchronization using corrected timestamps is challenging as it makes the model of the clock synchronization problem nonlinear.

This work introduces a new clock synchronization scheme, FLOPSYNC-3, that is capable of operating with timestamps corrected by the previous iteration of the algorithm itself. As a result of this improved capability, the Miosix RTOS was extended with a generic interface allowing to implement clock correction at the hardware timer level.

The FLOPSYNC-3 controller is here tested both in simulation and on a network of nodes running the Miosix operating system and the TDMH [20] real-time wireless mesh protocol.

This paper is organized as follows: Section 2 presents a brief overview on the state of the art in clock synchronization for distributed embedded systems. Section 3 discusses how the Miosix OS has been extended with a virtual clock abstraction that enables transparent clock corrections. Section 4 briefly mentions the design of the Miosix subsystem for performing timestamp measurements, a key feature used to precisely timestamp clock synchronization packets. Section 5 presents the FLOPSYNC-3 clock synchronization scheme that can perform clock corrections using the virtual clock as actuator while operating on corrected timestamps only. Finally, Section 6 presents simulation and experimental results, and Section 7 outlines future research directions.

2 Related Works

Clock synchronization is a classical problem in distributed systems [11, 15], but also one where research is still ongoing to produce clock synchronization schemes fine-tuned to changing application requirements and hardware capabilities. Many works related to clock synchronization in distributed embedded systems come from the Wireless Sensor Network research community, focusing on several aspects including low power synchronization [18, 22], propagation delay compensation [12, 19], efficient synchronization information dissemination [14, 8].

When considering accuracy, a major differentiating factor is whether a clock synchronization scheme only performs offset corrections or it also performs skew corrections. Simple schemes such as TPSN [9] and DMTS [17] only correct for offset. When implemented at the OS level, this correction can be efficiently performed by overwriting the hardware counter with the required correction at every synchronization [9]. The disadvantage is however that after each correction the hardware clock keeps counting at the incorrect frequency, and thus a time error accumulates over the synchronization period, which reaches its maximum value immediately before the next correction. Another issue is that the value returned by the clock exhibits a discontinuity at every synchronization [22], a matter that can introduce errors in interval measurements, especially for short intervals.

More advanced clock synchronization schemes perform skew (also known as frequency or rate) corrections. The synchronization scheme produces both an offset and a frequency correction value at every synchronization. As altering the frequency of a crystal oscillator requires additional hardware [5] which is usually unavailable in off-the-shelf boards, the frequency correction is preferably performed by applying an algorithm every time the OS or applications request the time. In this paper we refer to such algorithm as a virtual clock. For a given synchronization period, frequency correction allows for lower synchronization errors compared to offset correction. Additionally, clock synchronization schemes that perform frequency correction can make the corrected clock continuous and monotonic [22], thus avoiding clock jumps.

Real-time embedded systems also face increasing power and energy constraints [3], especially if battery operated. Clock synchronization may thus be required to operate also when the processor enters a deep sleep state which includes turning off the main oscillator. In such cases, time is kept using a low power Real-Time Clock (RTC), and this introduces the need to synchronize both the RTC and high-frequency timebase [21], a matter that we account for by designing our virtual clock to support multiple corrections.

In this paper we address the clock synchronization problem from the perspective of implementing it at the real-time OS level. Software engineering considerations suggest us to completely encapsulate time correction, and since this makes uncorrected time unavailable to the clock synchronization scheme, we design a new scheme that can operate with corrected timestamps.

3 Virtual Clock

A real-time OS typically requires two main time-related primitives: one to get the current time, whose use is obvious, and one to set an interrupt in a given future time instant, to be used to handle context switches as well as sleeping tasks wakeup. This chapter describes the design and implementation of a virtual clock to make these primitives synchronization-aware.

3.1 Design

An uncorected clock t_{nc} fed by an oscillator with nominal frequency f_0, affected by (possibly time-varying) frequency error δ_s will progressively diverge from an ideal one as

$$t_{nc}(t) = \int_0^t \frac{f_s(\tau)}{f_0} d\tau = t + \int_0^t \frac{\delta_s(\tau)}{f_0} d\tau \tag{1}$$

where f_s is the instantaneous oscillator frequency, and the integral accounts for the accumulated frequency error. Accordingly, the accumulated frequency error $\Delta(k)$ over one clock synchronization period k of duration T is

$$\Delta(k) = \int_{(k-1)T}^{kT} \frac{\delta_s \tau}{f_0} d\tau \tag{2}$$

A virtual clock VC is a piece wise linear function (Figure 1) that applied to the uncorrected clock t_{nc} produces a corrected one. Virtual clocks allow to perform not only offset corrections, but also frequency corrections. Said otherwise, it is possible to control a virtual clock to count time faster or slower than the underlying hardware clock to better approximate a reference clock. A virtual clock is however just an actuator, it provides the *means* to correct a hardware clock, but requires at every synchronization period updated parameters. A clock synchronization scheme uses a controller and time information from an external reference to adjust the virtual clock rate trying to align it to the reference clock. By defining the virtual clock rate separately on each synchronization interval, it can be demonstrated by induction that the value of the virtual clock (that is the corrected time t_c) on a generic time t_{nc} inside a synchronization interval $[\,kT, (k+1)\,T\,]$ can be expressed as

$$t_c = VC(t_{nc}) = VC(k) + \dot{VC}(k)\big(t_{nc} - t_{nc}(k)\big) \tag{3}$$

where $\dot{VC}(k)$ is the rate of the virtual clock. More specifically, if $t_{nc} = t_{nc}(k+1)$, its definition simplifies as

$$VC(k+1) = VC(k) + \dot{VC}(k)\,(T + \Delta\,(k)) \tag{4}$$

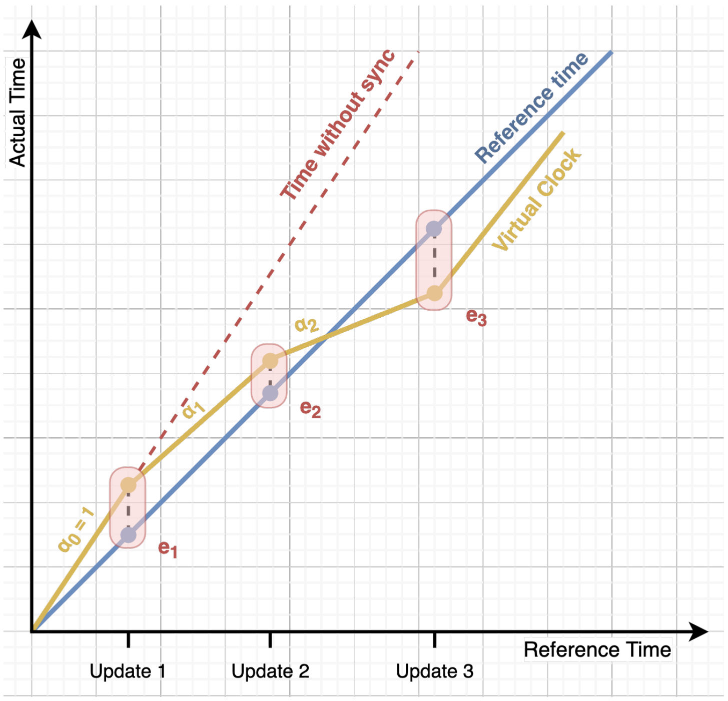

Figure 1 Virtual clock correcting clock rate to align itself to a reference clock.

We can further generalize the virtual clock expressing (3) as $f = a_k x + b_k$ by algebraic manipulation

$$\begin{cases} a_k & = \dot{VC}(k) \\ b_k & = VC(k) - \dot{VC}(k)t_{nc}(k) \end{cases} \tag{5}$$

where a_k is the rate correction of the clock in the synchronization period k and b_k is the offset. This rate is adjusted by the controller to align the current clock to the reference, and is related to the mean skew over the synchronization period.

3.2 Implementation

The virtual clock was implemented in C++ as part of the Miosix RTOS, as shown in Figure 2. To support clock synchronization as well as deep sleep operation which entails transitions from a board RTC to an high resolution clock (a technique called VHT [21]), a virtual clock may need to perform multiple clock corrections f_i combined. The software design of the virtual clock thus supports multiple corrections as a *Variable Length Correction Stack* (VLCS). This design allows for an arbitrary number of *Correction Tiles*, each with their own correction parameters $a_{k,i}$ and $b_{k,i}$. For performance reasons, the number of correction tiles is configured at compile time as a template parameter. Having n distinct corrections chained together as $f_0 \circ \cdots \circ f_n$, the combined correction parameters can be calculated as

$$a_{k,vc} = \prod_{i=0}^{n-1} a_{k,i} \tag{6}$$

$$b_{k,vc} = \sum_{i=0}^{n-2} \left\{ b_{k,i} \cdot \prod_{j=i+1}^{n-1} a_{k,j} \right\} + b_{k,n-1} \tag{7}$$

where index 0 is the correction closer to the hardware timer, and n the furthest.

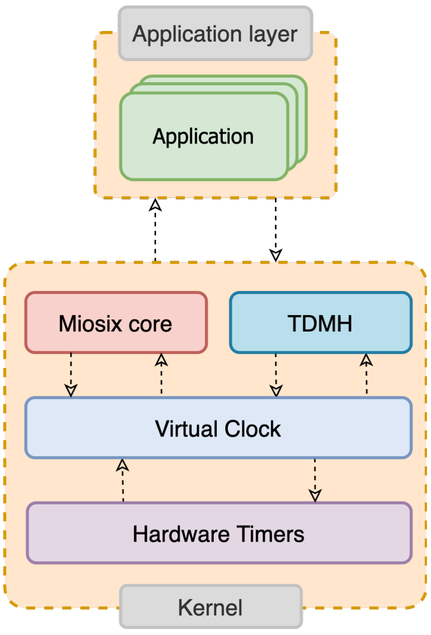

Figure 2 Virtual clock interface.

As reading the current time is a more frequent operation than changing the correction coefficients, the combined parameters are precomputed when a new clock correction is produced (Figure 3). Conversely, to set a time interrupt the corrected time coming from the OS will need to be back-converted as the hardware timer still works using uncorrected time.

$$t_c = a_{k,vc} \cdot t_{nc} + b_{k,vc} \tag{8}$$

$$t_{nc} = (t_c - b_{k,vc}) \, / \, a_{k,vc} \tag{9}$$

3.3 Optimization

As the typical skew of quartz clocks is in the order of tens of parts per million (*ppm*), the $a_{k,vc}$ coefficient should be very close to 1. Since many microcontrollers lack a Floating Point Unit (FPU), we need an efficient way (exploiting the range of $a_{k,vc}$ as just identified) to

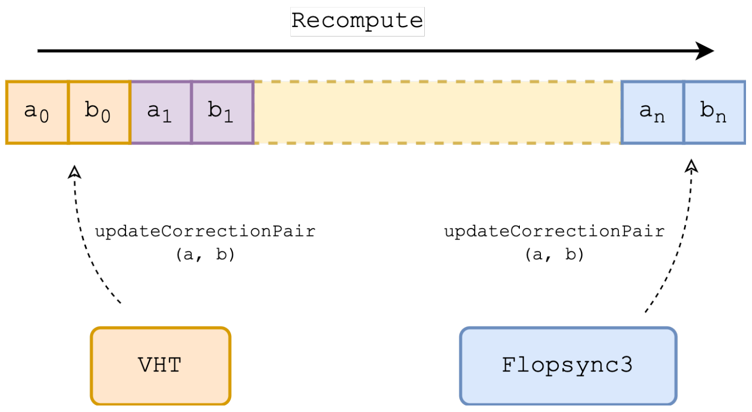

Figure 3 Virtual clock recomputing aggregated parameters a_{vc} and b_{vc}.

perform the multiplication $a_{k,vc} \cdot x$, as the time retrieval is one of the most critical path of the operating system. For this purpose, a template class *Fixed* was designed. This is capable of representing a fixed point number with an arbitrary number of bits for the decimal part. Given a few compile-time optimized functions able to handle multiplication between a 64-bit integer and a fixed point 32.32, a specialization of the said class – called *fp32_32* and representing a fixed point number as a 64-bit integer with 32-bit for both decimal and integer part – was used. With fp32_32, the multiplication $a_{k,vc} \cdot x$ can be performed in just 60 clock cycles bringing the total time to get the current time to 170 clock cycles, a 37% improvement compared to the previous implementation. Regarding the uncorrection, we can note from (9) that a division by $a_{k,vc}$ is needed. There is no nice properties to perform fast division using fixed point, so it was implemented as a multiplication for the inverse. The inverse value is precomputed using 64-bit floating point numbers at every update of the $a_{k,vc}$ parameter and converted to fp32_32. The pre-computation is optimized using a modified version of the fast inverse square root algorithm [6], adapted to perform $1/x$ instead of $1/\sqrt{x}$ as follows. The optimization relies on the fact that an *IEEE754* double precision number is very similar to an Logarithmic Number System (LNS) number, as they never differ for more than a small factor. An interesting property of LNS numbers is that it makes implementations for multiplications and divisions very efficient. In particular, the inverse of an LNS number v is $-v$. The bit representation of a floating point number u can approximated as the LNS number $x = 2^{u/2^{52}-1023}$, and using this representation the inverse q can be computed as follows

$$2^{q/2^{52}-1023} = 2^{-(u/2^{52}-1023)} \tag{10}$$

which solving the implicit equation results in

$$q = \text{0x7FE0000000000000} - u \tag{11}$$

Performing a sweep with sufficient precision, it was possible to elaborate a quadratic regression model to approximate faster and with more precision the hexadecimal value. Having a closer approximation, less Newton steps are necessary making the inversion faster. This whole inversion process is called *optimizedFastInverse*.

4 Hardware Events

Although what presented above is sufficent to support the time-related requirements of an OS, performing clock synchronization requires accurate timestamping of received radio packets. Moreover, advanced radio transmission techniques such as constructive interference require accurate packet transmission times [22]. To abstract hardware-accelerated event timestamping and generation, Miosix was extended with an *Eventstamping* interface. This abstraction introduces the concept of *event channels* that abstract the event sources or sinks of a given platform. Every event channel can be configured as *input*, for external event timestamping, or as *output* to trigger events. When configured in input mode, a thread can block and wait for an event to happen on the chosen channel, with an optional timeout (Figure 4). When configured as output, a thread can generate a hardware event in the future, blocking until that time point. This design simplifies the realization of TDMA networking protocols. Since events are measured/generated in hardware, the achievable time granularty is that of the hardware timer (in our implementation 21ns), and is unaffected by software interrupt latencies.

5 FLOPSYNC-3

The redesign of the Miosix OS timing subsystem in order to only operate in terms of corrected time in the entire OS –except for the timer driver– required the design of a new clock synchronization scheme. Previously, the clock synchronization packets were timestamped using the uncorrected clock, as this was needed by the FLOPSYNC-2 algorithm [22]. The previous approach required to deal with both corrected and uncorrected times and was causing code maintainability issues from a software engineering standpoint. However, performing clock synchronization using timestamps corrected by the previous round of clock synchronization makes the problem nonlinear. The FLOPSYNC-3 controller was designed to address the aforementioned nonlinearity, and implemented at the OS level.

5.1 Design

Given (4), we can define the clock synchronization error at the end of each synchronization period as

$$e(k) = VC(k) - kT \tag{12}$$

To observe the evolution of the error across synchroniation periods, we can compute the next error as a function of the previous, resulting in

$$e(k+1) = e(k) + T\left(1 - \overset{\bullet}{VC}(k)\right) - \Delta(k)\overset{\bullet}{VC}(k) \tag{13}$$

where the $\Delta(k)\overset{\bullet}{VC}(k)$ term makes the model nonlinear. To perform the control synthesis we used *Feedback Linearization* [4] to linearize this process using the output $u(k)$ of a linear controller and express the new error as

$$e(k+1) = \beta\, e(k) + (1-\beta)\, u(k) \tag{14}$$

and as a consequence have the new output $u(k)$ of the controller provide $\overset{\bullet}{VC}(k)$ from

$$\overset{\bullet}{VC}(k) = \frac{e(k)(1-\beta) + u(k)(\beta - 1) + T}{T + \Delta(k)}. \tag{15}$$

The mean skew value at the synchronization period k is of course not available and needs to be approximated with the previous one $(k-1)$, i.e.,

$$\hat{\Delta}(k) = \Delta(k-1) = \frac{VC(k) - VC(k-1)}{\alpha(k-1)} - T \tag{16}$$

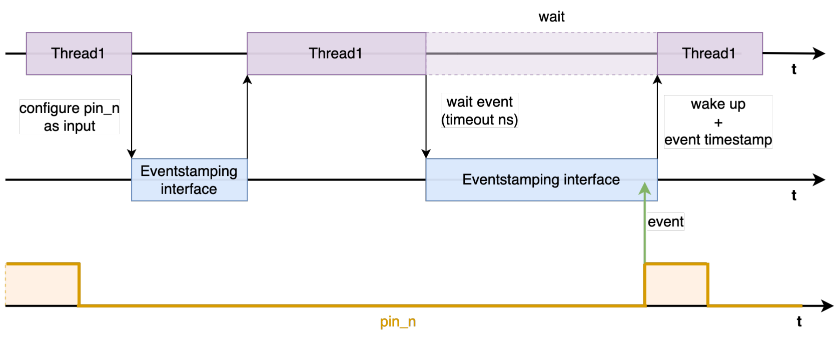

Figure 4 Eventstamping, wait event.

We can then obtain the transfer function $\mathscr{H}(z)$ of the imposed dynamic (14) as

$$\mathscr{H}(z) = \frac{E(z)}{U(z)} \Rightarrow \frac{1-\beta}{z-\beta} \tag{17}$$

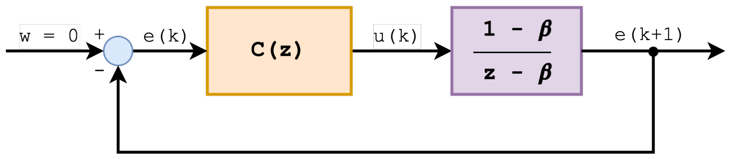

Figure 5 FLOPSYNC-3 Control scheme.

The controller $C(z)$ used to work in conjuction with the feedback linearization is a proportional one having a gain of 0.15. The full FLOPSYNC-3 control scheme is shown in Figure 5.

6 Simulation and Experimental Results

The operation of the FLOPSYNC-3 controller and virtual clock were first assessed through simulations performed using the *Modelica* language.

Figure 6 shows one such simulation, where the clock synchronization period T was set to 10 seconds and β was chosen to be 0.025. The left part of the figure shows the simulated clock skew profile, that starts from 10 ppm and increases to 50 ppm from $T = 150$ seconds, approximating in the simulation the effect of an ambient temperature change. The right part of the figure shows the clock synchronization error. The blue line is the instantaneous error of the virtual clock, thus the time error exposed to the operating system and application. As a node in the network can measure its error only at discrete intervals, corresponding to when synchronization packets are received, the red line shows the measured error that feeds the FLOPSYNC-3 controller.

As can be seen, the initial 10 ppm skew causes a 100 μs error that is quickly corrected by the FLOPSYNC-3 controller. The frequency change caused by the simulated temperature change, although higher in amplitude than the initial skew causes a lower peak error, less than 75 μs, due to its slower nature.

The FLOPSYNC-3 controller as specified by equation (15) and (16) has been implemented in Miosix acting on the the variable length correction stack of the virtual clock. Since deep sleep support was not implemented, the correction stack was configured to perform the FLOPSYNC-3 correction only. Synchronization parameters T and β were configured as in the simulations. The clock synchronization error measurement was taken from the TDMH networking stack using the eventstamping interface to provide the timestamps of received synchronization packets. FLOPSYNC-3 was implemented using the fp32_32 type to perform intermediate calculations efficiently. Because of the limited range of this type, pre-scaling was necessary to avoid overflows.

Clock synchronization experiments were performed with a network of nodes running the TDMH networking stack on top of Miosix. Figure 7 shows the clock synchronization error of one such node. The top part of the figure shows the measured clock synchronization error in the first three minutes after synchronization. The initial clock synchronization error of 40 μs occurs when the node is booted and joins the network. This value is the accumulation of the oscillator frequency error over the entire first synchronization period, as the FLOPSYNC-3 algorithm, being a feedback one, requires a first error measure to compute a correction. The

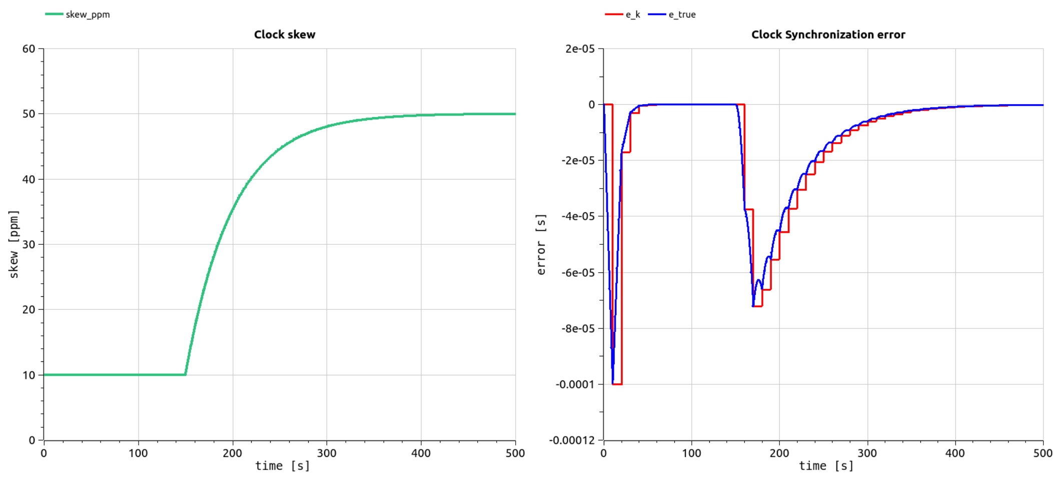

Figure 6 Simulated clock skew profile (left) and clock synchronization error (right). The blue line shows the instantaneous synchronization error, while the red line shows the measured error.

bottom part of the figure shows the error after the initial synchronization, over a period of approximately 24 hours, to better appreciate the error dynamics after the initial skew is corrected. The observed stochastic nature of the clock synchronization error, not present in the simulations, is caused by the measurement noise of packet timestamps. The error standard deviation, excluding the first transient, is 137 ns.

7 Conclusions

This work addressed abstracting clock synchronization at the operating system level. To achieve this goal a virtual clock was introduced as an efficient abstraction allowing a hardware timer driver to provide a time reference whose rate can be changed compared to the one of the underlying oscillator. Support for multiple corrections sources was accounted for, allowing the implementation of deep sleep solution such as VHT [21]. Encapsulating time correction allows reducing bugs and problems during development since all components are just using the same time source (corrected), but makes the uncorrected synchronization packet timestamps unavailable to the clock synchronization algorithm. The FLOSPYNC-3 controller was thus designed specifically to overcome this issue.

The Miosix real-time OS was extended with a flexible, efficient and modular timing subsystem based on the virtual clock design, capable of internalizing the clock correction and only exposing corrected time to all kernel and application tasks. This new timing subsystem was designed from the start to be general allowing to easily port the Miosix to different microcontrollers.

Future research directions will focus on further improving clock synchronization resilience to temperature variations, while future improvements of the Miosix timing subsystem will address completing the support for maintaining clock synchronization during deep sleep periods using the variable length correction stack.

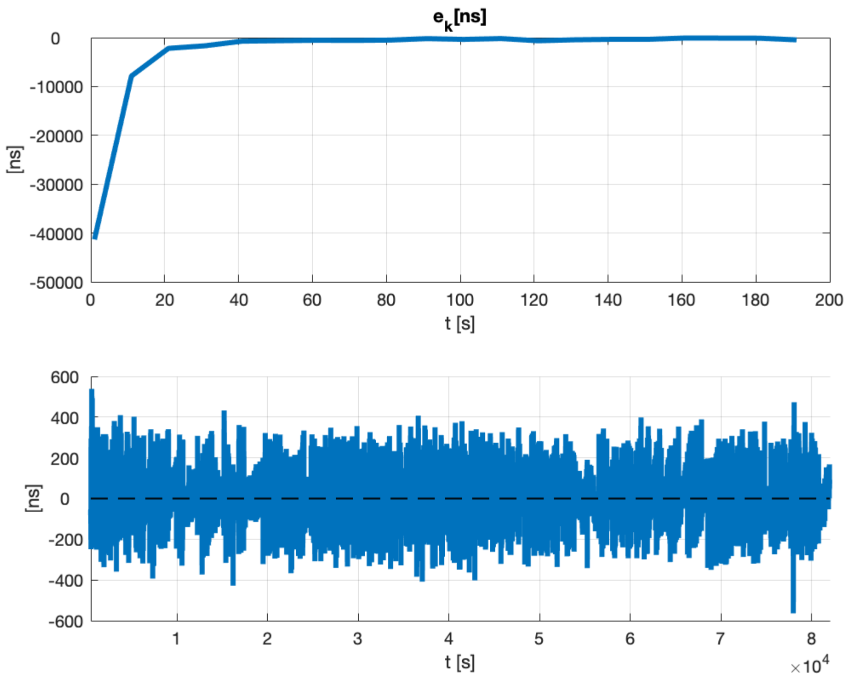

Figure 7 Clock synchronization error during experimental evaluation. Top plot includes the initial clock skew compensation, bottom plot shows the synchronization error after the initial transient.

References

1 Diogo Almeida, Miguel Gaitán, Pedro d'Orey, Pedro Santos, Luis Ramos Pinto, and Luís Almeida. Demonstrating RA-TDMAs+ for robust communication in WiFi mesh networks. In *RTSS@work workshop co-located with the 42nd IEEE Real-Time Systems Symposium*, 2021.

2 Riad Azzam and Nabil Aouf. Visual information to enhance time difference of arrival based acoustic localization. In *2014 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology*, pages 77–82, 2014. doi:10.1109/ICARES.2014.7024400.

3 Ashikahmed Bhuiyan, Federico Reghenzani, William Fornaciari, and Zhishan Guo. Optimizing energy in non-preemptive mixed-criticality scheduling by exploiting probabilistic information. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3906–3917, 2020. doi:10.1109/TCAD.2020.3012231.

4 Roger W Brockett. Feedback invariants for nonlinear systems. *IFAC Proceedings Volumes*, 11(1):1115–1120, 1978.

5 Maxim Buevich, Niranjini Rajagopal, and Anthony Rowe. Hardware assisted clock synchronization for real-time sensor networks. In *2013 IEEE 34th Real-Time Systems Symposium*, pages 268–277, 2013. doi:10.1109/RTSS.2013.34.

6 John Carmack. Fast inverse square root. URL: https://blog.timhutt.co.uk/fast-inverse-square-root/.

7 Julius Degesys, Ian Rose, Ankit Patel, and Radhika Nagpal. DESYNC: Self-Organizing Desynchronization and TDMA on Wireless Sensor Networks. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks*, pages 11–20, 2007. doi:10.1145/1236360.1236363.

8 F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *Information Processing in Sensor Networks (IPSN)*, 2011.

9 S. Ganeriwal, R. Kumar, and M. Srivastava. Timing-sync protocol for sensor networks. In *International Conference on Embedded Networked Sensor Systems*, 2003.

10 Grzegorz Krukar, Marco Wenzel, Piotr Karbownik, Norbert Franke, and Thomas von der Grün. Proof-of-concept real time localization system based on the UWB and the WSN technologies. In *2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 756–757, 2014. `doi:10.1109/IPIN.2014.7275559`.

11 L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 1978.

12 Roman Lim, Balz Maag, and Lothar Thiele. Time-of-flight aware time synchronization for wireless embedded systems. In *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, EWSN '16, pages 149–158, USA, 2016. Junction Publishing.

13 L. Maillet and C. Fraboul. Scheduling complex real-time tasks in an embedded distributed system. In *Proceedings Seventh Euromicro Workshop on Real-Time Systems*, pages 62–65, 1995. `doi:10.1109/EMWRTS.1995.514293`.

14 M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. The flooding time synchronization protocol. In *Conference On Embedded Networked Sensor Systems*, 2004.

15 D.L. Mills. Internet time synchronization: the network time protocol. *IEEE Trans. on Communications*, 39(10):1482–1493, 1991.

16 Dinh C. Nguyen, Ming Ding, Pubudu N. Pathirana, Aruna Seneviratne, Jun Li, Dusit Niyato, Octavia Dobre, and H. Vincent Poor. 6G Internet of Things: A Comprehensive Survey. *IEEE Internet of Things Journal*, 9(1):359–383, 2022. `doi:10.1109/JIOT.2021.3103320`.

17 Su Ping. Delay measurement time synchronization for wireless sensor networks. In *Intel Research*, 2003.

18 A. Rowe, V. Gupta, and R. Rajkumar. Low-power clock synchronization using electromagnetic energy radiating from ac power lines. In *Sensys*, pages 211–224, 2009.

19 Federico Terraneo, Alberto Leva, Silvano Seva, Martina Maggio, and Alessandro Vittorio Papadopoulos. Reverse Flooding: Exploiting Radio Interference for Efficient Propagation Delay Compensation in WSN Clock Synchronization. In *2015 IEEE Real-Time Systems Symposium*, pages 175–184, 2015. `doi:10.1109/RTSS.2015.24`.

20 Federico Terraneo, Paolo Polidori, Alberto Leva, and William Fornaciari. TDMH-MAC: Real-Time and Multi-hop in the Same Wireless MAC. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 277–287, 2018. `doi:10.1109/RTSS.2018.00044`.

21 Federico Terraneo, Fabiano Riccardi, and Alberto Leva. Jitter-Compensated VHT and Its Application to WSN Clock Synchronization. In *2017 IEEE Real-Time Systems Symposium (RTSS)*, pages 277–286, 2017. `doi:10.1109/RTSS.2017.00033`.

22 Federico Terraneo, Luigi Rinaldi, Martina Maggio, Alessandro Vittorio Papadopoulos, and Alberto Leva. FLOPSYNC-2: Efficient Monotonic Clock Synchronisation. In *2014 IEEE Real-Time Systems Symposium*, pages 11–20, 2014. `doi:10.1109/RTSS.2014.14`.

23 Hüseyin Yiğitler, Behnam Badihi, and Riku Jäntti. Overview of time synchronization for iot deployments: Clock discipline algorithms and protocols. *Sensors (Switzerland)*, 20(20):1–58, 2020. `doi:10.3390/s20205928`.